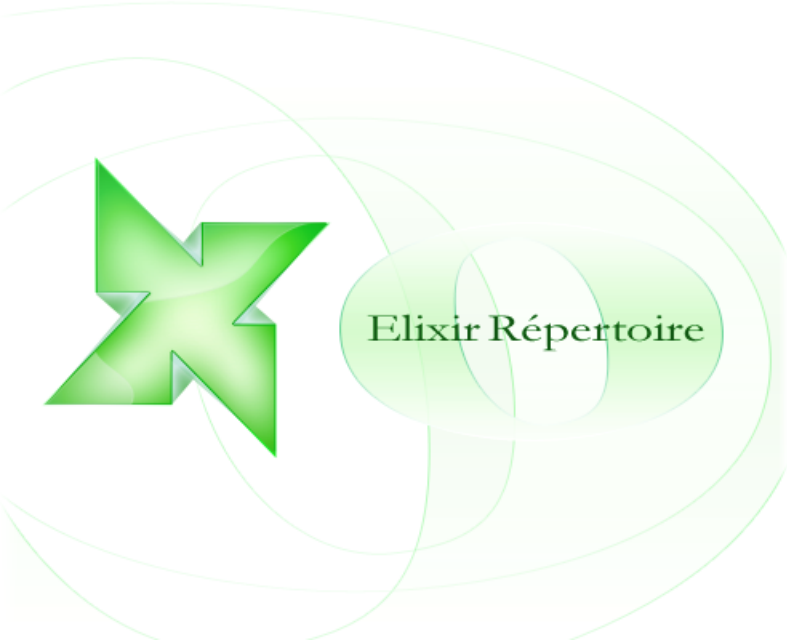
Elixir Dashboard Designer User Manual

Release 7.3



Elixir Technology Pte Ltd

Elixir Dashboard Designer User Manual: Release 7.3

Elixir Technology Pte Ltd

Published 2008
Copyright © 2005-2008 Elixir Technology Pte Ltd

All rights reserved.

Solaris, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. Microsoft and Windows are trademarks of Microsoft Corporation.

Table of Contents

1. About Elixir Dashboard Designer 1
    Introduction 1
        Contents 2
        Views 2
        Actions 2
    The Dashboard Toolbar 2
    Dashboard Dimension 3
    Case Study 3
2. Contents 9
    Introduction 9
    Dynamic DataSources 9
    Filtering, Sorting and Extracting 11
    Presentation 12
    Initial Selection 13
    Chart 14
    Cube Table 15
    Glint 15
    Grid 16
    Image 18
    Report 19
    Table 20
    Case Study 1 21
    Case Study 2 23
3. Views 25
    Introduction 25
    Card 25
        Layout 25
    Split View 26
    Content View 26
    Maximize 26
    Drag and Drop 27
4. Actions 28
    Introduction 28
    Content Events 28
    Available Actions 29
        Maximize Action 29
        Refresh Action 29
        Show Action 29
        Show Card Action 29
        Reset Action 29
        Render Report Action 29
        Generate Data Action 29
        Generate Excel Cube Action 29
        Script Action 30
    Triggering Actions from Events 30
    Case Study 1 30
    Case Study 2 31
5. Script Reference 34
    Introduction 34
    Parameters 34
    State 34
    Individual States 35
Index 38

List of Figures

1.1. A Sample Dashboard 1
1.2. Cards, Views and Contents 2
1.3. The Dashboard Toolbar 2
1.4. Dashboard Dimension 3
1.5. Table on Perspective 4
1.6. Placement of Intended Content 5
1.7. Chart and Table on Perspective 6
1.8. Grid, Chart and Table on Perspective 7
1.9. Completed Perspective 8
2.1. *View* URL 9
2.2. Dynamic Datasource 10
2.3. Datasource Parameters 11
2.4. The W3C CSS Box Model 13
2.5. Initial Selection Dialog 13
2.6. Chart on Perspective 14
2.7. Cube Table on Perspective 15
2.8. Glint on Perspective 16
2.9. Add Content Wizard 17
2.10. Grid on Perspective 18
2.11. Image on Perspective 19
2.12. Report on Perspective 20
2.13. Table on Perspective 21
2.14. Initial Selection Wizard 22
2.15. Results of Case Study 23
2.16. Results of Case Study 2 24
3.1. Layout 25
3.2. Split View 26
4.1. Default screen of Action Wizard 31
4.2. Generate Data Wizard 31
4.3. Sample 32
4.4. Render Wizard 33
5.1. Script Action 34

Chapter 1
About Elixir Dashboard Designer

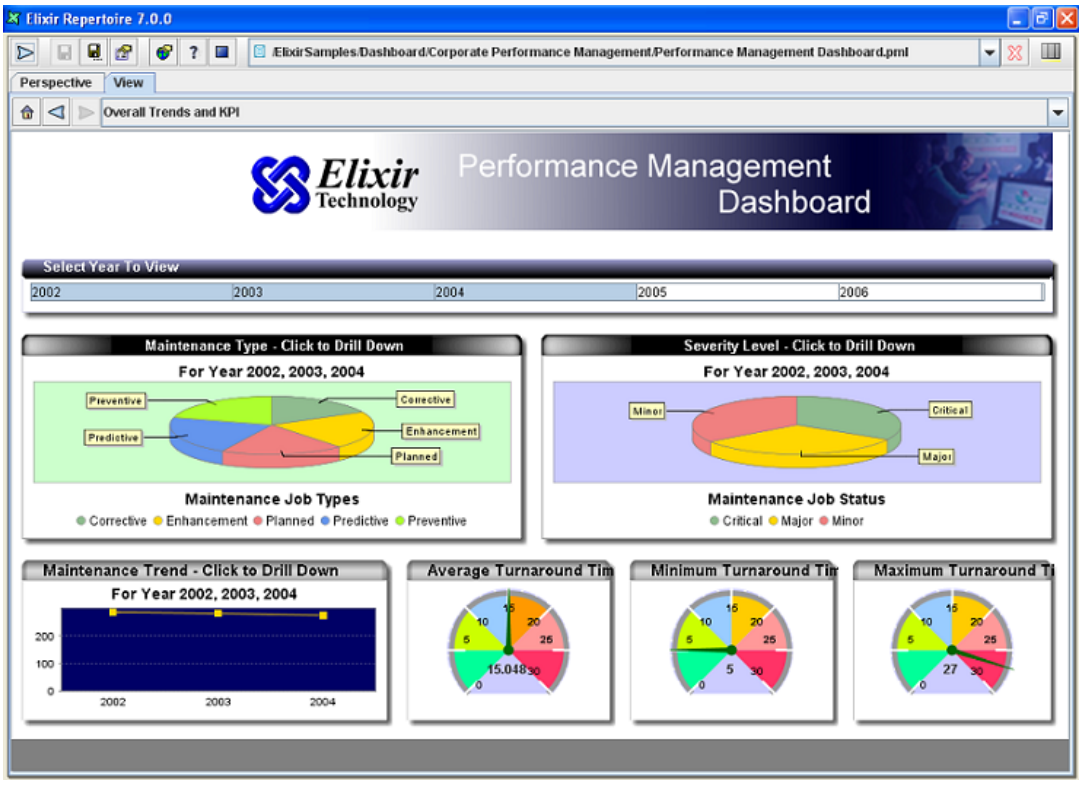
Introduction

Elixir Dashboard Designer is a tool for navigating and visualizing enterprise data. A dashboard can assemble a comprehensive view of the available information. It provides:

- Seamless navigation between data, cube, and reports with user-customizable views
- On-demand cube creation and modification of dimensions and measures for data analysis
- Graphical visualization of data and OLAP cube
- User-definable linking and filtering of data from disparate data sources
- Dynamic report generation from view selections

A Perspective is the set of views, contents and actions that together form a dashboard. Dashboards are stored in files with the extension `.pml`, which stands for Perspective Markup Language.

Figure 1.1. A Sample Dashboard



To achieve this powerful and flexible solution, Elixir Dashboard Designer defines Content, Views and Actions.

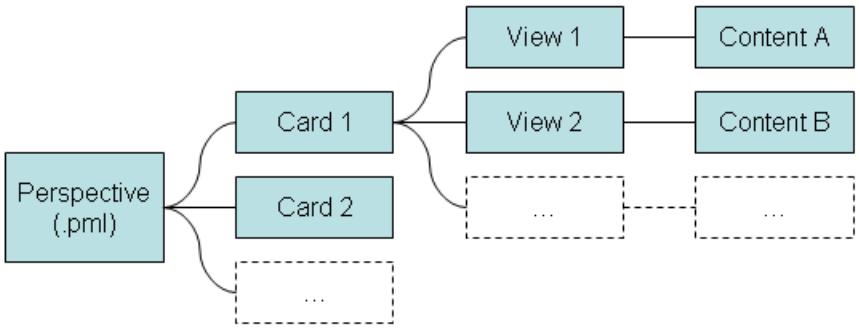
Contents

The content is the underlying data and presentation. This could be a report or cube backed by a data source, or a chart showing data derived from other Contents. Each content added to a Perspective has a unique name allowing it to be referenced by views. The different kinds of content supported by in a dashboard are described in Chapter 2, *Contents*.

Views

A view acts as a frame for content. Content may appear in different views at different times. Views may have titlebars that allow interaction with the underlying content and act as placeholders for connecting content together. For example, a chart may receive it's data for rendering from the selection in another view. All views also have unique names to allow this kind of referencing. Views are grouped together into cards. Elixir Dashboard Designer shows one card (containing one or more views) at a time and user actions can be used to trigger navigation between cards. The different kinds of view supported by dashboards are described in Chapter 3, *Views*.

Figure 1.2. Cards, Views and Contents



Actions

Actions are invoked in response to events . Typically these are user interaction events, like selecting a row in a table, or clicking on a detail in a report. Certain actions are preconfigured - if a chart shows the data from a table and the table contents changes, then the chart is automatically refreshed. Other actions can be user-defined. For example, double-clicking on a chart may open another view on that data. Actions are chosen based on a combination of the user event, the view and the content involved, so a double-click on two charts may result in two completely different actions - it is completely user-definable. The different kinds of actions supported by a dashboard are described in Chapter 4, *Actions*.

The Dashboard Toolbar

Figure 1.3. The Dashboard Toolbar

The Elixir Dashboard Designer toolbar is shown in Figure 1.3, “The Dashboard Toolbar”. From left to right, the contents are:

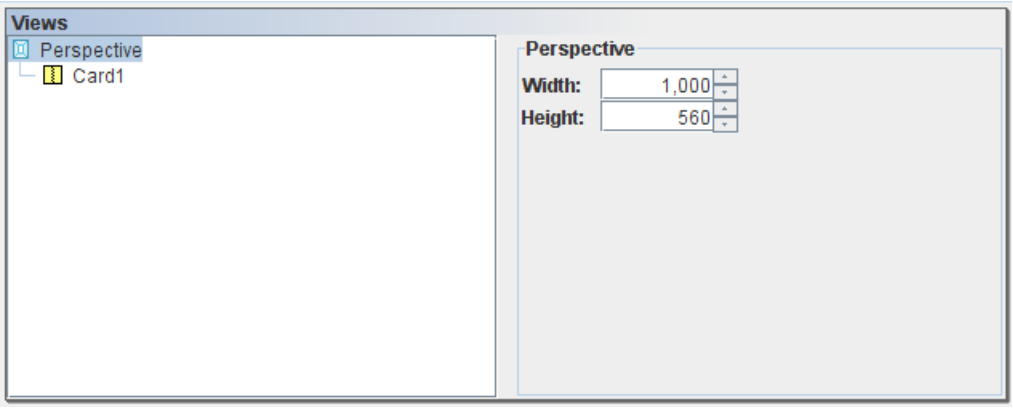
Home — This button resets the dashboard to it's initial state. That is, all views are restored to their initial contents and initial selection.

Back	Just like a browser, this button reverts to the previous dashboard state. This can undo any undesirable change, or navigate back to a previous position to allow further exploration.
Forward	Again, just like a browser, this button takes you forward again, redoing any action that was undone by pressing Back.
Card Combo	This combo box shows all of the cards in the dashboard and allows you to choose one to view. Actions can also be used to change the card as a result of events.

Dashboard Dimension

The dimension of a dashboard can be specified by selecting the Perspective root of the view tree as shown in Figure 1.4, “Dashboard Dimension”. The dimensions are measured in pixels.

Figure 1.4. Dashboard Dimension

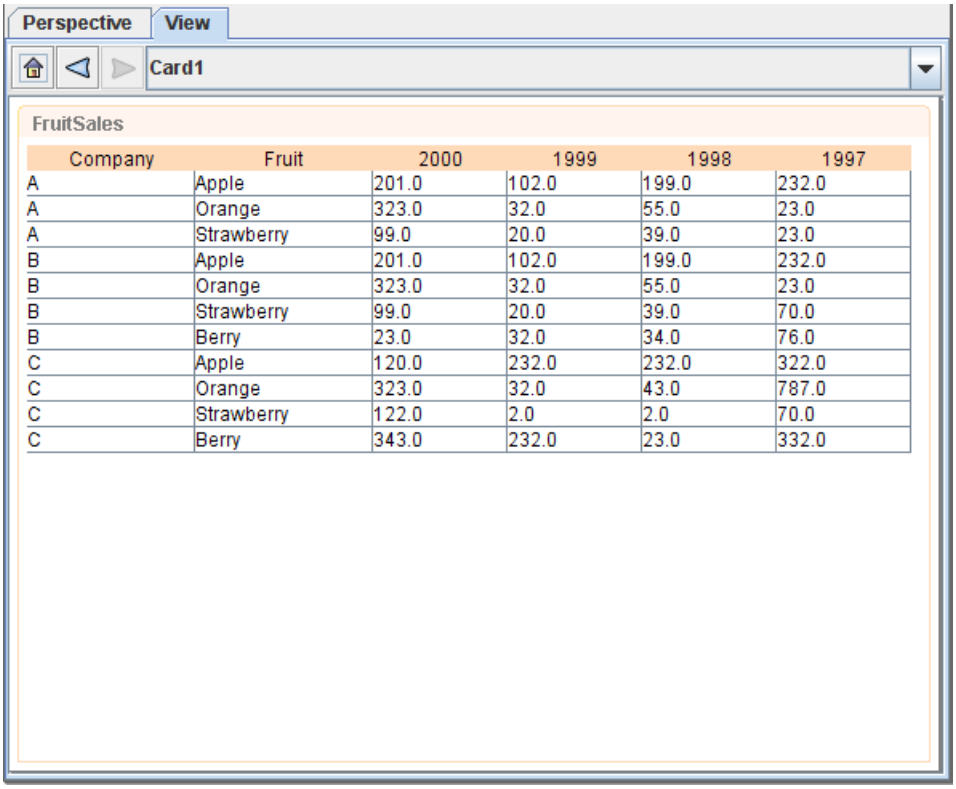


Case Study

This case study will create a simple dashboard built upon the /ElixirSamples/DataSource/FruitSales.ds datasource.

1. Select a FileSystem. Right-click to select Add->Perspective from the filesystem popup menu.
2. When the Add Perspective window appears, enter a unique name for the Perspective and click Finish.
3. Drag FruitSales.ds from the repository to the workspace. On releasing of the mouse, a popup with four different options will appear. It's for the user to select the element to add to the Perspective. In this case, we will add a Table element by selecting `Add Table`.
4. On clicking, a table with all the data from FruitSales.ds will be displayed. The Table element will appear.

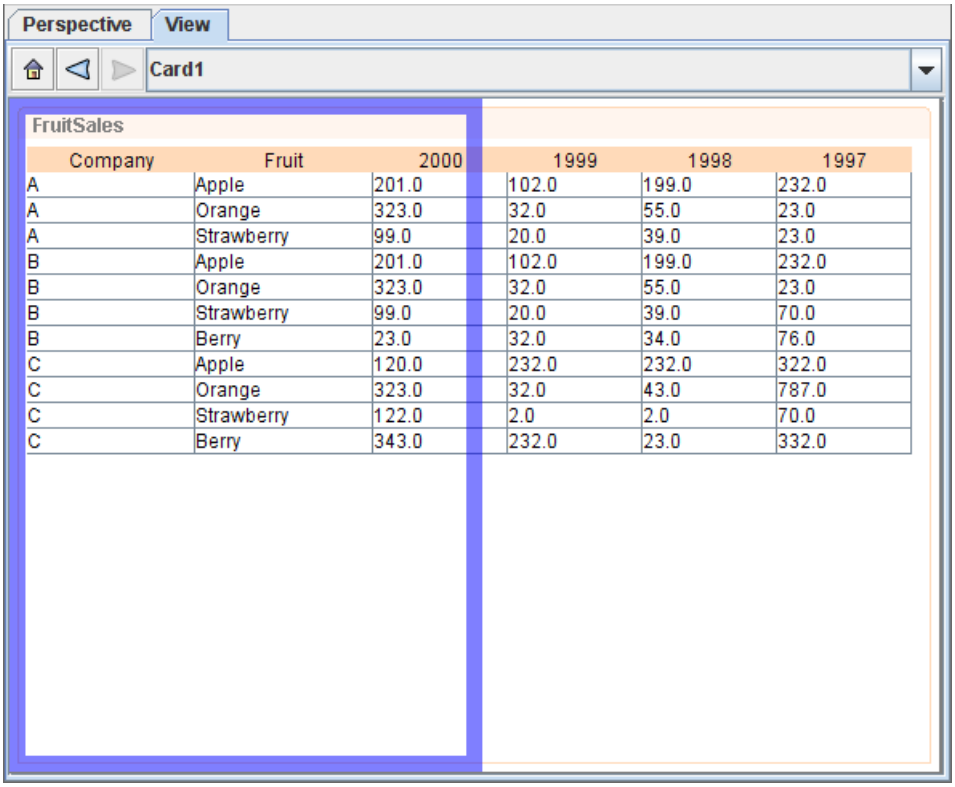
Figure 1.5. Table on Perspective



Company	Fruit	2000	1999	1998	1997
A	Apple	201.0	102.0	199.0	232.0
A	Orange	323.0	32.0	55.0	23.0
A	Strawberry	99.0	20.0	39.0	23.0
B	Apple	201.0	102.0	199.0	232.0
B	Orange	323.0	32.0	55.0	23.0
B	Strawberry	99.0	20.0	39.0	70.0
B	Berry	23.0	32.0	34.0	76.0
C	Apple	120.0	232.0	232.0	322.0
C	Orange	323.0	32.0	43.0	787.0
C	Strawberry	122.0	2.0	2.0	70.0
C	Berry	343.0	232.0	23.0	332.0

5. Drag FruitSales.ds into the workspace again. A blue box will be shown to indicate the placement of intended content, as shown in Figure 1.6, “Placement of Intended Content”. Over here, we will be placing it on the left of the table. When the mouse-click is released, select `Add Chart` and the content will be created on that region.

Figure 1.6. Placement of Intended Content

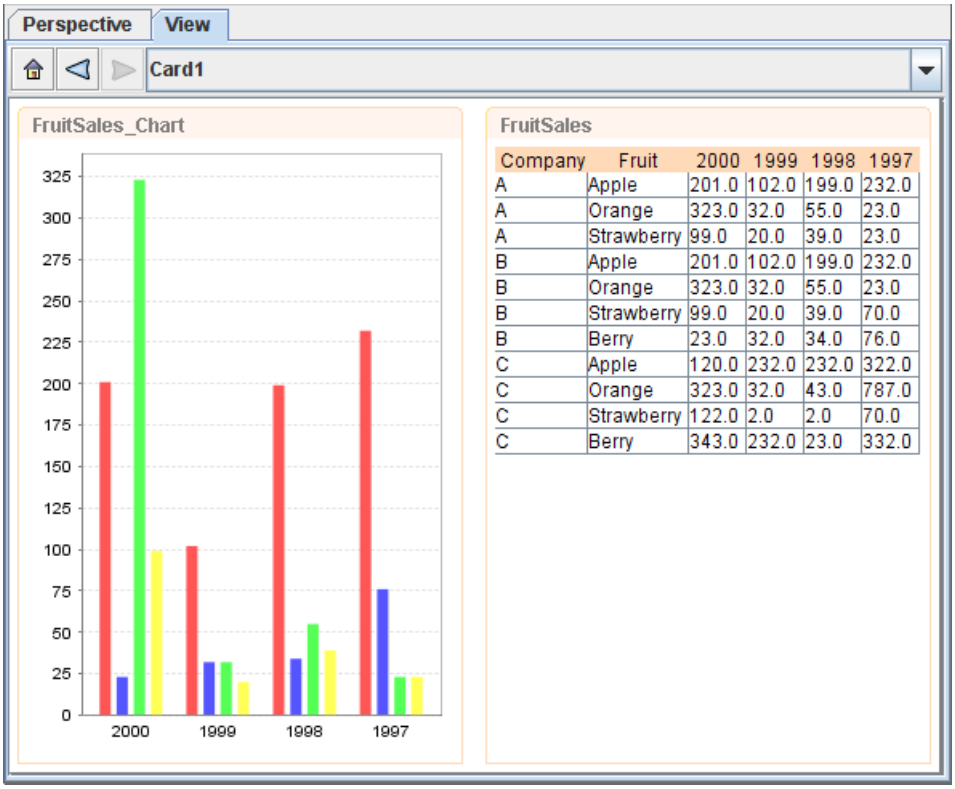


Company	Fruit	2000	1999	1998	1997
A	Apple	201.0	102.0	199.0	232.0
A	Orange	323.0	32.0	55.0	23.0
A	Strawberry	99.0	20.0	39.0	23.0
B	Apple	201.0	102.0	199.0	232.0
B	Orange	323.0	32.0	55.0	23.0
B	Strawberry	99.0	20.0	39.0	70.0
B	Berry	23.0	32.0	34.0	76.0
C	Apple	120.0	232.0	232.0	322.0
C	Orange	323.0	32.0	43.0	787.0
C	Strawberry	122.0	2.0	2.0	70.0
C	Berry	343.0	232.0	23.0	332.0

6. In the *Add Content Wizard*, change the name to "FruitSales_Chart" for easy reference.

7. In this example, we will create a Column Chart with *Fruit* as the key and *1997*, *1998*, *1999* and *2000* as the values.

Figure 1.7. Chart and Table on Perspective



Company	Fruit	2000	1999	1998	1997
A	Apple	201.0	102.0	199.0	232.0
A	Orange	323.0	32.0	55.0	23.0
A	Strawberry	99.0	20.0	39.0	23.0
B	Apple	201.0	102.0	199.0	232.0
B	Orange	323.0	32.0	55.0	23.0
B	Strawberry	99.0	20.0	39.0	70.0
B	Berry	23.0	32.0	34.0	76.0
C	Apple	120.0	232.0	232.0	322.0
C	Orange	323.0	32.0	43.0	787.0
C	Strawberry	122.0	2.0	2.0	70.0
C	Berry	343.0	232.0	23.0	332.0

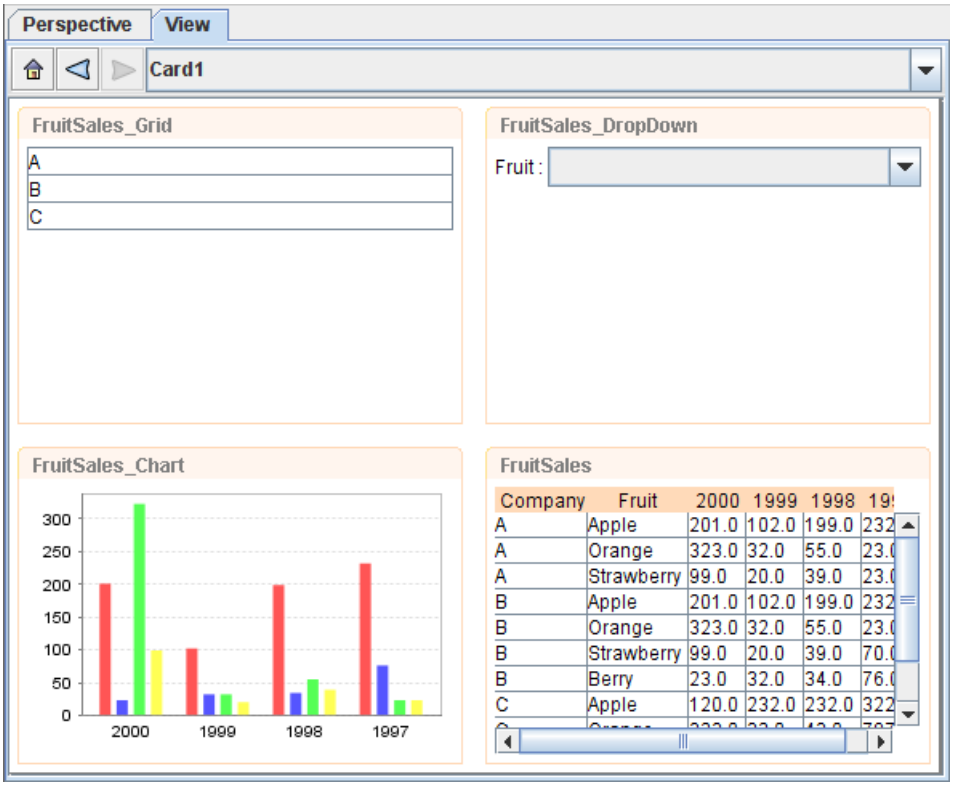
8. Now we will try creating a Grid element in the Perspective. Repeat step 1 to 3, but place the element on top of *FruitSales_Chart*, select `Add Grid` and change the name to *FruitSales_Grid*.

9. At the next page of the wizard, select a field for the Grid element to display. Select *Company* then click on `Finish`.

10. Repeat step 9 and 10, place the element above *FruitSales* and change the name to *FruitSales_DropDown*.

11. In the *Add Content Wizard*, select *Fruit* as the field and check the checkbox that says *Show as Drop-Down List*. In the `Label` field, enter *Fruit :* .

Figure 1.8. Grid, Chart and Table on Perspective



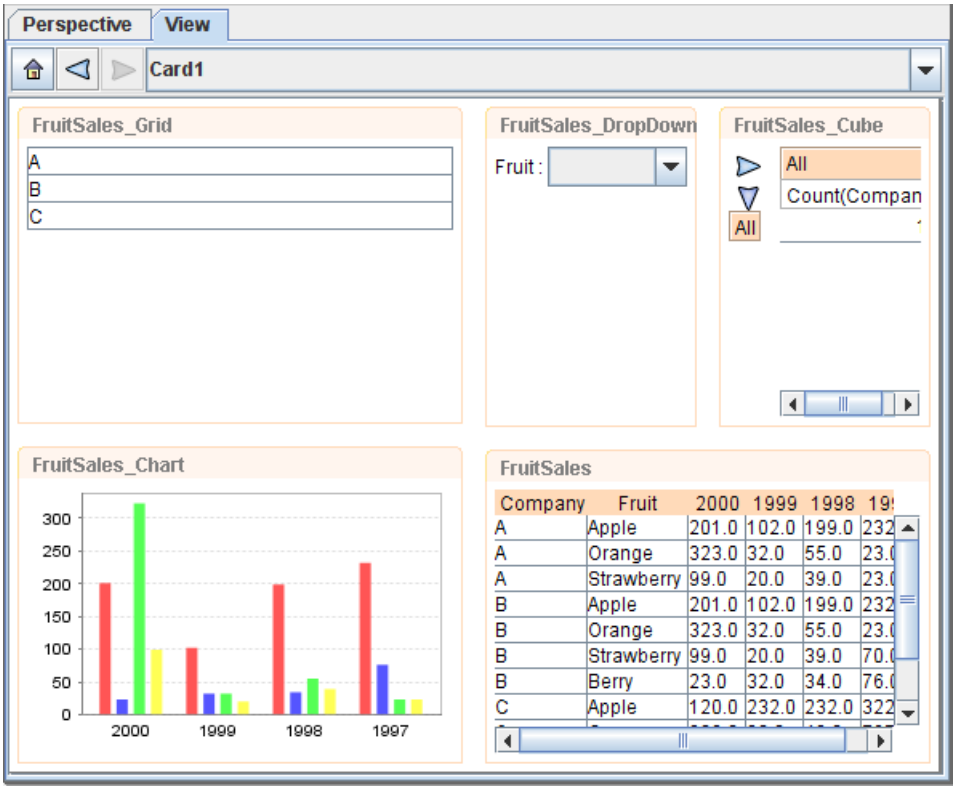
12. Repeat step 1 and 2. Place the blue region on the right of *FruitSales_DropDown* and select `Add Cube`.

13. Click through the wizard without changing any settings.

14. You can adjust the height and width of each element by dragging the sides.

15. At the end of this, the whole Perspective should look something as shown in Figure 1.9, “Completed Perspective”

Figure 1.9. Completed Perspective



Chapter 2
Contents

Introduction

Elixir Dashboard Designer provides different kinds of content from which you can compose your views. Every content must have a unique name within the Perspective so that it can be referenced. Each kind of content reads data and renders it into a view. The views then provide the ability to interact with the content.

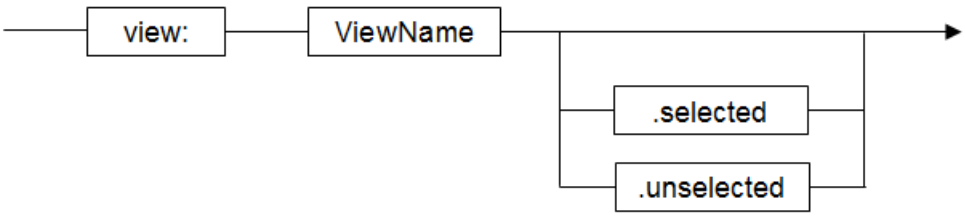
Most kinds of content read from a datasource. There are two kinds of datasource available, the .ds files in the repository (which can be JDBC, XML, etc.) and dynamic datasources where the data records are extracted from other views. The first kind of datasource will be identified with a `repository:` URL, whereas the dynamic data will be accessed with a `view:` URL. When the datasource is defined in the repository, a schema is associated with it and is available to the content designer. When the datasource is defined by a view, there is no explicit schema available - it depends on the content of the view, which can itself be dynamic. The solution is to allow view-based datasources to explicitly load a schema. When a datasource has been chosen for a content, the subsequent page in the wizard will show the schema of that datasource if one is available. If no schema is available, the `Load Schema` button will be available to allow one to be explicitly loaded.

The data schema needs to be available in order to support options like filtering. Each content that reads from a datasource is allowed to filter the records before rendering them. The filtering options allow records to be selected based on criteria such as "Equals","Greater Than","In Range","In DataSet" etc. These are discussed in the Filter Processor description of the Elixir Data Designer manual.

Dynamic DataSources

Dynamic datasources allow a content to read it's data from another view, rather than from a fixed datasource. This allows the user to select an item in one view and see related items appear automatically in another view. A dynamic datasource is identified by a URL with the protocol `view:`. The next part of the URL is the name of the view. For example, `view:Top-Left` would use the data supplied by the view called Top-Left. Many views support selection and it is often useful to show information related to the selected records of another view. This can be accomplished by adding the suffix `.selected` onto the URL. Similarly, `.unselected` will use those records that are not selected. Taking the previous example, the URL `view:Top-Left.selected` would get the selected data records only from the Top-Left view.

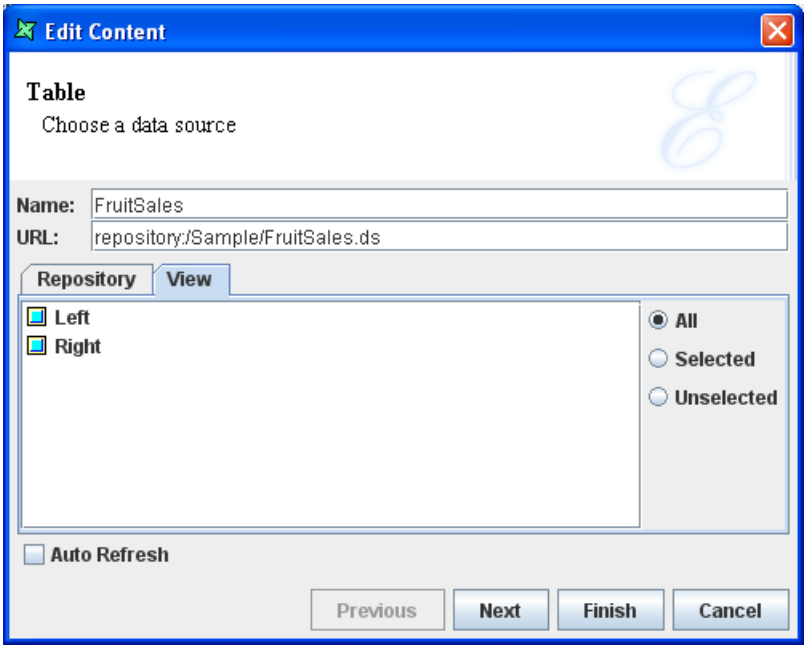
Figure 2.1. *View* URL



The tool will manage the connection between views so that any view that depends on another is automatically updated when the dependent view changes. There is usually no need to define explicit actions to enable this behaviour. If you use scripts to extend the dependency mechanism - for example changing the dependency dynamically - then you can explicitly define the dependency on the last page of the content wizard. It is important to note that datasources can only be read from views, not from contents. This is because each content may be filtered and selected differently within different views at the same time. The benefit of selecting through views is that the same view may show different contents at different times and the dynamic data connection is still maintained.

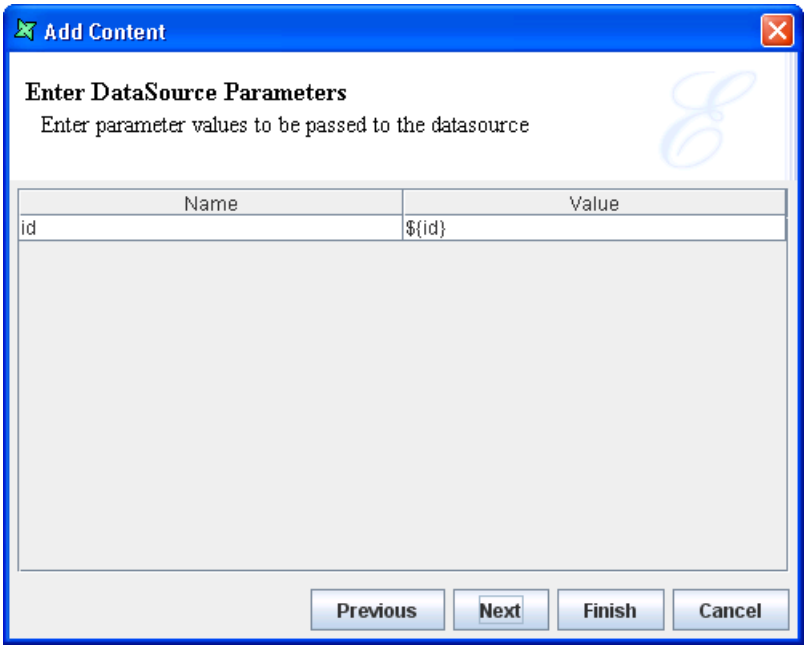
For each kind of content, described below, there is an indication of **Available datasources:** that indicates what options each supports. Support for All, means the view name can be used to get the entire datasource of the view. Support for Selected means the view name followed by .selected can be used to get the selected records from the view. Support for Unselected means the view name followed by .unselected can be used to get the unselected records from the view. The figure below, Figure 2.2, “Dynamic Datasource”, shows the screen where a user selects a dynamic datasource to use.

Figure 2.2. Dynamic Datasource



If a datasource has a parameter, there will be a page in the wizard for specifying the value of the parameter as shown in Figure 2.3, “Datasource Parameters”.

Figure 2.3. Datasource Parameters



Filtering, Sorting and Extracting

Data from a datasource or another view can be filtered and sorted before being passed to the view. Certain records can also be extracted, for example the top ten or bottom five can be retained and the rest discarded (this extraction operation occurs after any filtering and sorting).

Filtering allows you to define rules that each record is tested against. A typical rule might be: Country Equals "Singapore" or Salary Greater Than 2000. Only those records that pass all the rules are allowed to proceed. The filter criteria used here are the same as those used in the Filter Processor in the Data Designer manual. Please see that discussion for full details on the options available.

Sorting allows you to sort the records into ascending or descending order based on the values in fields. Multi-level sorting is supported, so you can sort by Country and then City and the tool will ensure that the Cities are sorted within their respective Countries. Certain views (such as Grid and Chart) also support sorting. Sorting within the view will override any earlier sorting, but will occur after any extraction. For example, you can sort sales people by performance, extract the top ten and then show them in a chart sorted in alphabetical order. Two variants of sort are supported, regular Ascending and Descending and Simple Ascending and Simple Descending. The Simple options are very fast, but don't take into account complex character differences, for example letters with accents produced by merged Unicode character combinations. The regular Ascending and Descending use the full Unicode algorithms. Further, the Simple variants treat whitespace as significant, whereas the Unicode algorithms don't (just like a dictionary).

Extraction is the retention of certain records based on their position in the data set. You might choose to extract those records at the top or the bottom of the set (presumably the order is significant, probably because they have just been sorted). You can choose to retain either a fixed number or a percentage of all the records. If a percent value is chosen, then the value will be "rounded up" to include partial records. For example, if you choose 1%, you will still get one record, even if there are only twenty records in the set.

Presentation

Each kind of content has some common presentation characteristics, along with some content-specific ones. These are all editable from the Presentation Wizard, accessible from the Presentation popup menu item. The common characteristics are:

Padding — Padding is the area inside the border, which provides insets for the content area. The background of the padding is set according to the background colour of the content. See Figure 2.4, “The W3C CSS Box Model”.

Margins — Margins is the area outside the border, which provides insets for the border rectangle. The background of the margins is set according to the background colour of the card that the content is shown on. See Figure 2.4, “The W3C CSS Box Model”.

Background — The background colour is used to fill the background of the border and padding rectangles, along with the content area. If no background is specified then usually the card background will show though (this depends on the content type and other content-specific backgrounds).

Title Bar Tip — Title Bar Tip is a field for entering some text for the tooltip. The tooltip will appear when a cursor is placed over the top bar of the element's content border. You will need to enter some text under Title Bar Tip and apply border to the element before any tooltip can be seen.

Border Name — A Border is a set of images in a repository folder that are used to construct a frame around the content (and padding). To select a border, you need to select the folder that contains the images. There are a number of samples provided in /ElixirSamples/Resources/Borders, for example you might choose /ElixirSamples/Resources/Borders/Lavender. You can create your own custom borders by following the examples. Your borders don't need to have images on all sides of the frame. Any missing images will just appear blank. It is important for correct web display that the widths and heights of all images are consistent. For example the heights of topleft, top and topright images must be the identical.

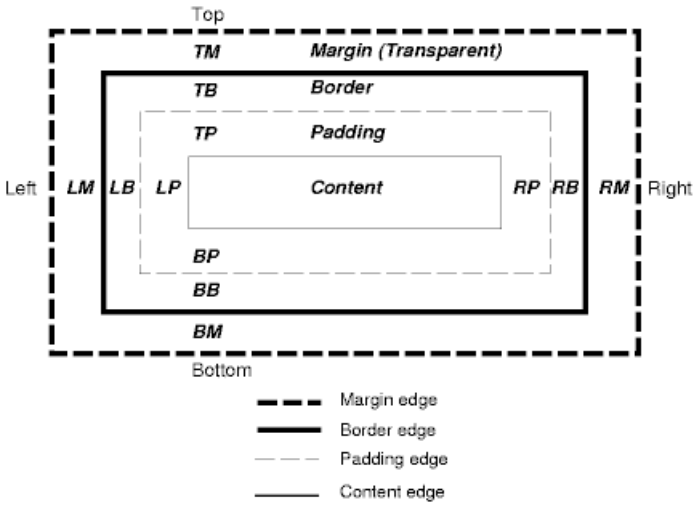
Border Title — If a border is chosen that supports titles (if there is a title.xml in the border folder), then any text entered here will be placed on the border using the information provided by the title.xml. You may either use a fixed text string, or substitutions. The available substitutions are:

${ContentName}	the name of the content
${ViewName}	the name of the view
${ShortDate}	the short form of the current date
${ShortTime}	the short form of the current time
${=JavaScript}	Any JavaScript (after an equals (=) will be evaluated at runtime and the result substituted.
${view:*ViewName.Selection:Field*}	The field name of the selected/unselected record in a particular view. However, the element must be refreshed before being able to show the updated border title. An ideal way of utilising this feature will be using on an element that displays data based on the selected/unselected of a

view. This will be discussed later in the case study.

You can mix these, for example "${ContentName} : ${ShortDate}" will show both the content name and short date in the title. For other titles, you can use script substitutions, like this: ${=java.lang.System.getProperty("java.version")}. Any JavaScript syntax (which can call Java, as shown) inside a ${=JavaScript} substitution will be evaluated when the title is shown.

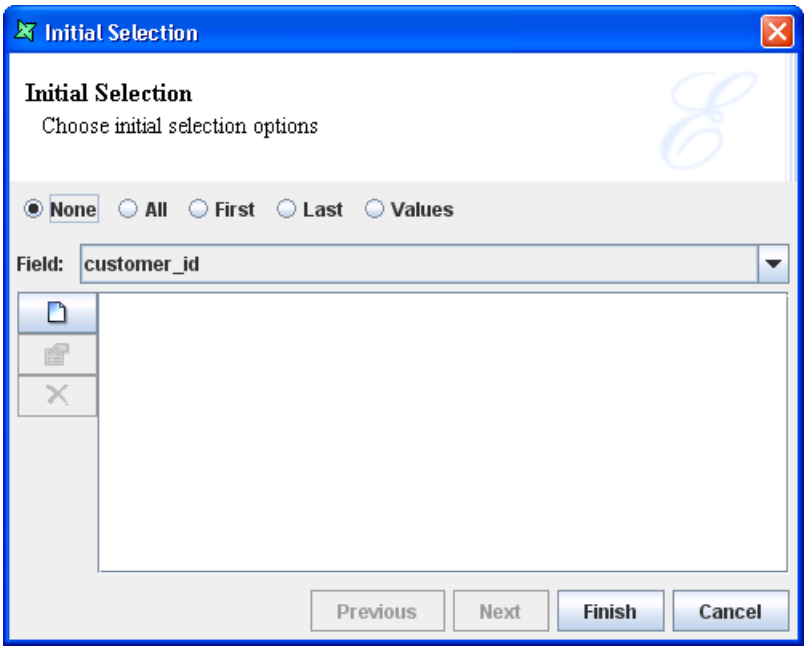
Figure 2.4. The W3C CSS Box Model



Initial Selection

When the dashboard is first loaded, it is often useful for contents to be given initial selections, so that the user isn't presented with a blank page. No initial selection is provided by default, but you can use the Initial Selection dialog, available on the popup menu of each content (or if you prefer, each view of the content). The dialog is shown in Figure 2.5, “Initial Selection Dialog”.

Figure 2.5. Initial Selection Dialog



The choices for initial selection are:

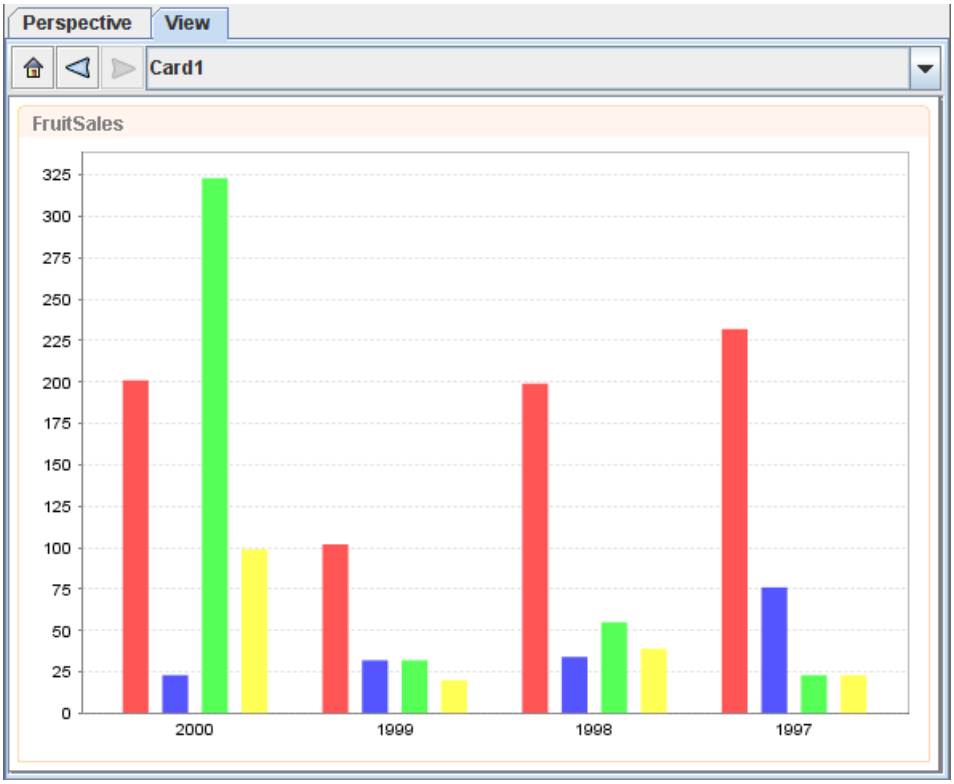
None	The default - no items are initially selected.
All	All items are initially selected.
First	The first item is selected (depending on sort order, if any).
Last	The last item is selected (depending on sort order, if any).
Values	This option requires a field name and one or more values to compare against the records. All records that contain values of the chosen field that match the value strings will be selected. Note that values are strings, so any other data types will be formatted into strings if necessary, for comparison purposes.

Chart

A chart reads data from a datasource and renders it as a graphical display which sizes to fill the view it is placed in. The datasource chosen may be identified by a repository URL, in which case the content is fixed, or the datasource may be reading from another view, in which case the chart will be updated automatically each time the other view is modified.

There are many parameters that can be used to configure the presentation of the chart, most importantly what type of chart is required. The chart wizard and range of available charts and options is described in the Elixir Report Designer manual.

Figure 2.6. Chart on Perspective



Bar, column, area and pie charts have selectable regions corresponding to the chart shapes. For example, clicking on a bar will select all records that contributed to that bar. You can create another view which uses this selection as a filter to control its display. You can also attach actions, so that clicking on a bar will show a new view or card which is mapped to the selection made. This is an easy method of creating a drill-down chart.

Available datasources: All, Selected, Unselected (only for bar, column, area and pie charts).

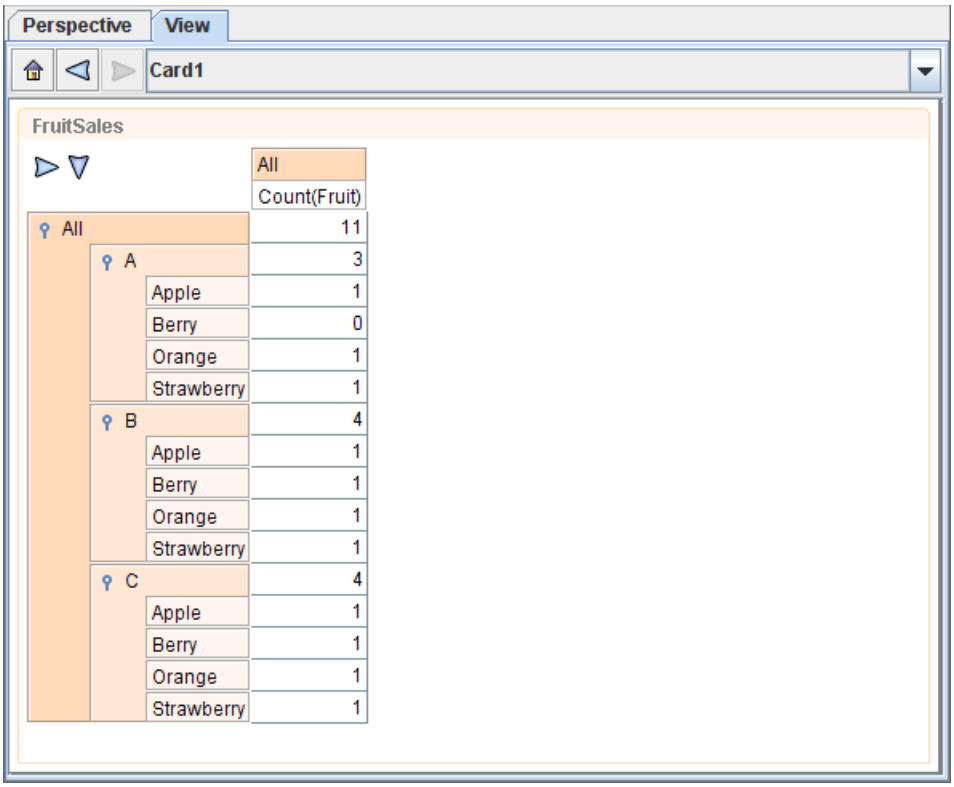
Cube Table

A cube table reads data from a datasource and renders a two-dimensional table showing the values derived from those records that form the intersection of the two dimensions. There are two ways in which a cube table can be designed. The Cube Table wizard provides a step-by-step dialog to guide the design process. This is identical to the wizard used in Elixir Report Designer and is described in that manual. The other approach is to create an empty cube table using the wizard (just click `Finish` after choosing a datasource) and designing the cube interactively. This approach is usually preferred because each change to the cube is immediately reflected in the view.

If the cube data is required for subsequent processing, it is possible to save the cube table contents into a CSV, DataSource, Excel or XML file.

Available datasources: All, Selected, Unselected.

Figure 2.7. Cube Table on Perspective



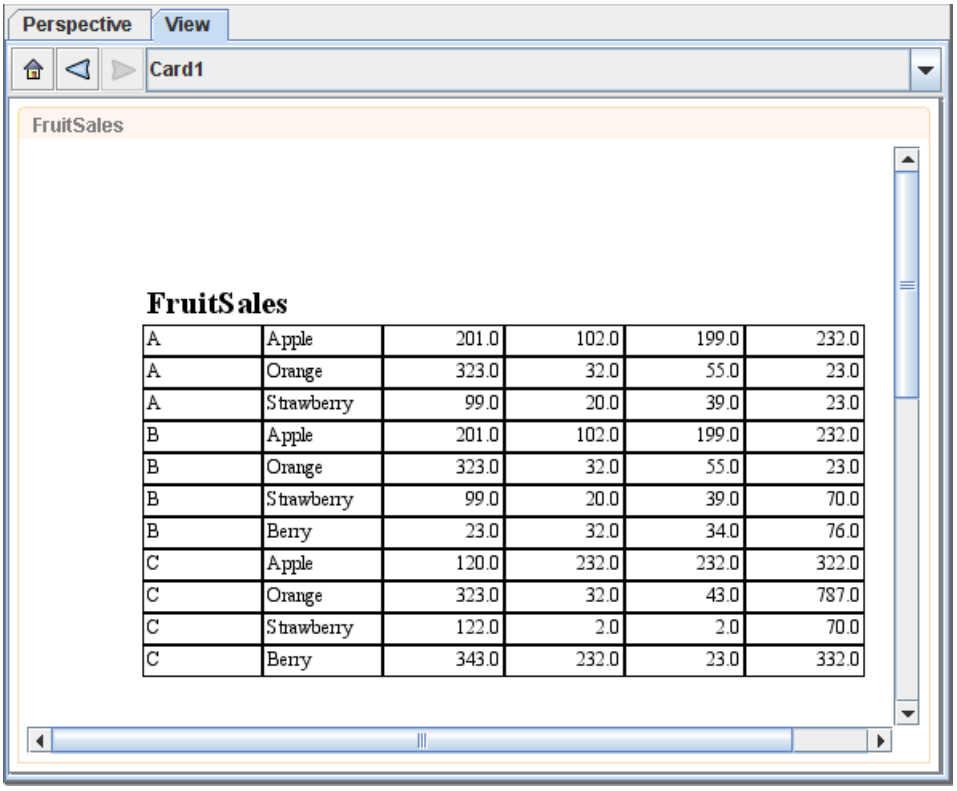
Glint

The glint content type renders a glint file to produce a graphical view. Glint is Elixir's native vector graphics format. It is designed to be extremely compact to minimize network traffic and reduce memory requirements on low-end machines. Instructions on generating a glint file can be found in the Report Designer documentation.

This is useful for displaying fixed content and pre-rendered output. For example, a view could be based on a FileSystem DataSource to show a list of glint files. Selecting a file would load it and display it in another view, allowing quick creation of a customized interface to access pre-rendered reports.

Available datasources: None.

Figure 2.8. Glint on Perspective



Grid

The grid content type is a special kind of table. The grid renders one field from each record and allows for a more compact representation, where the fields may be displayed in a grid layout.

Before rendering, any duplicate fields are removed and the values can optionally be sorted. When you select a field in the grid, you are actually selecting all records in the datasource that have that selected value. For example, with a grid showing the Field "Month", selecting "April" will select all records where Month=April.

When designing a grid, you can choose a direction, `Across, then Down` or `Down, then Across`, and either length in that direction. For example, a "4 Across, then Down" grid will place the cells like this:

```
1  2  3  4
5  6  7
```

There will be four cells in each horizontal row, and the fields will be added across then down. Alternatively, a "3 Down, then Across" grid would look like this:

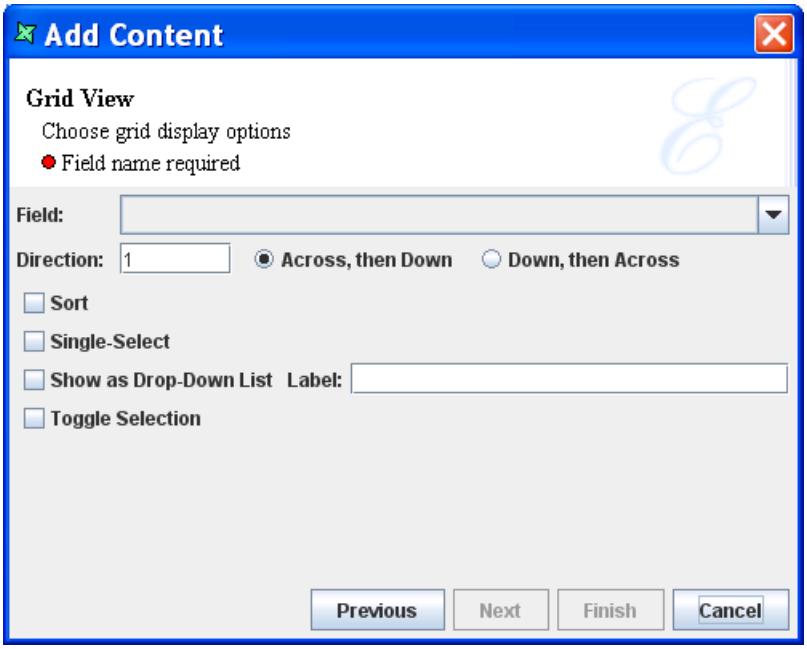
```
1  4  7
2  5
3  6
```

In this case there are four cells vertically and the fields are added down then across.

A grid can also be displayed as a Drop-Down list. This can be done on the second page of the Add Content wizard. The second page of Add Content wizard is shown in Figure 2.9, “Add Content Wizard”. The normal behaviour of grid selection is to turn off other selections each time a new selection is made (unless Control is held down). The grid also includes a Toggle Selection option which changes the behaviour so that the first click selects an item and a subsequent click on the same item is required to deselect it. This behaviour is often preferred as it prevents accidentally losing a set of existing selections by forgetting to press Control.

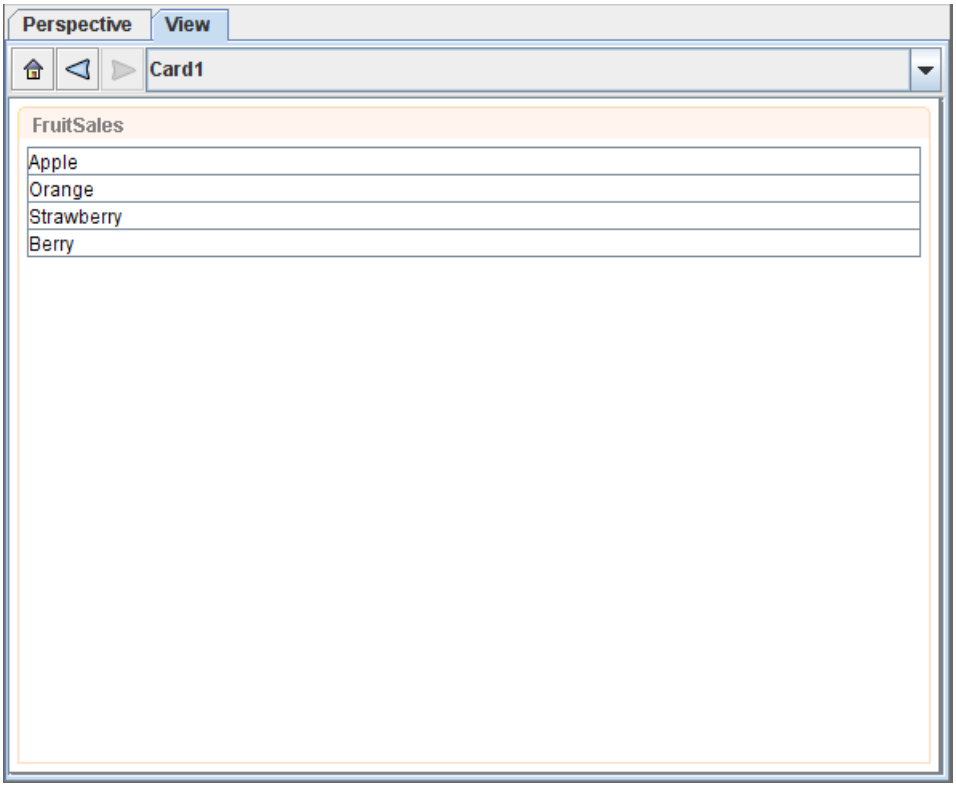
Check on the checkbox that says "Show as Drop-Down List" to create a Drop-Down list. Enter a name for the Label so a name will be displayed beside the Drop-Down list in the Dashboard. See Figure 2.9, “Add Content Wizard”. Note that the *Sort* in Figure 2.9, “Add Content Wizard” overrides the *Sort* in the earlier page of the wizard.

Figure 2.9. Add Content Wizard



Available datasources: All, Selected, Unselected.

Figure 2.10. Grid on Perspective

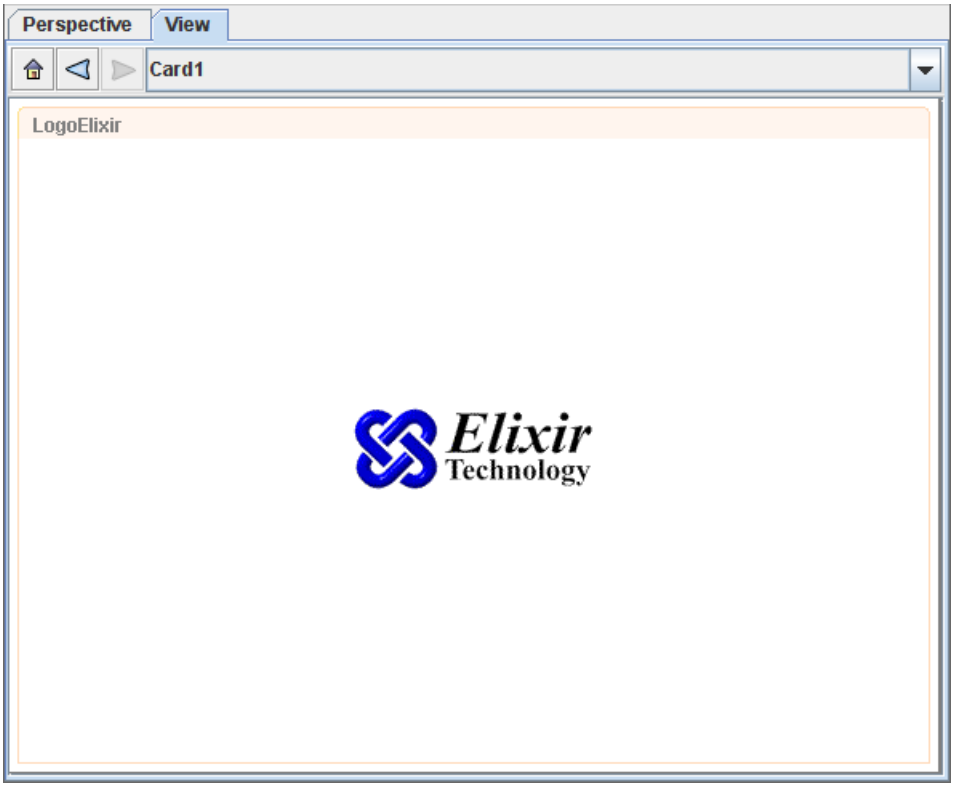


Image

An Image element is a rectangular region which picture files can be loaded into. Elixir Dashboard Designer supports image formats such as jpg, png and gif. As jpg and png file formats are supported across all implementations, they will be a best choice. Bmp and tiff image formats are not supported for Elixir Dashboard Designer.

Available datasources: None.

Figure 2.11. Image on Perspective

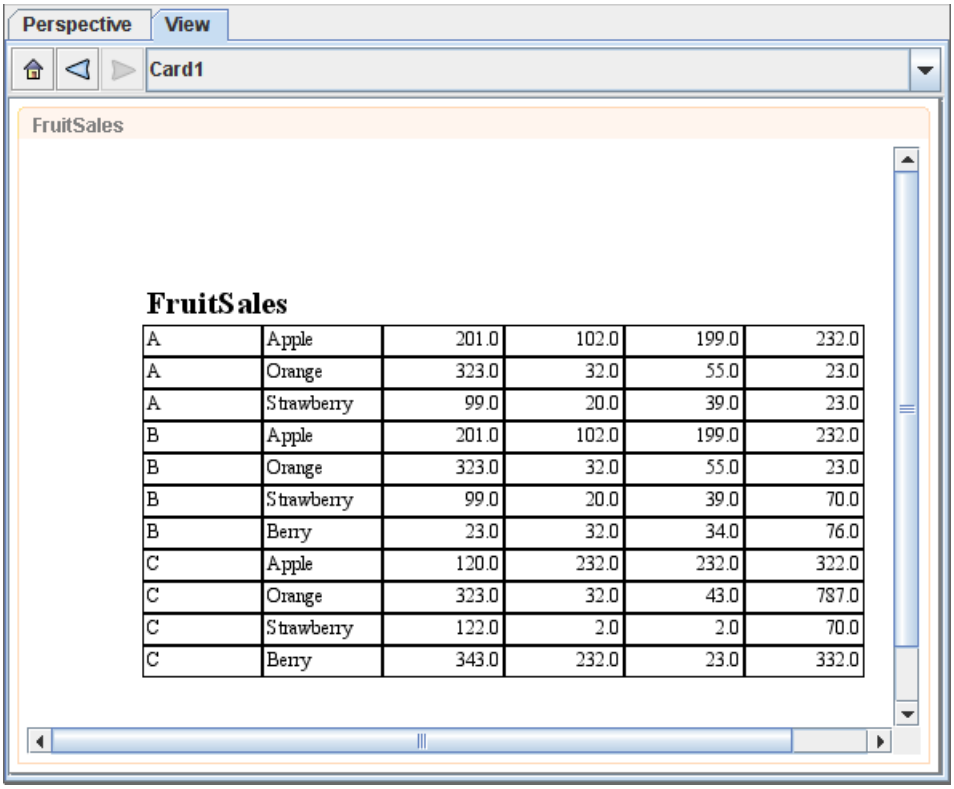


Report

Report templates created by Elixir Report Designer can be shown within a view. By default the templates will be rendered with their original datasources, as if rendered within Elixir Report Designer itself. However, you can also substitute new datasources derived from dynamic views in the Perspective. This allows interactive selection and filtering of data values for reporting. The report can be saved into any Elixir Report Designer output format, including Glint, PDF and HTML. Note that within a Perspective, a report is always shown in Streamed mode (no page breaks, page headers or footers). When the report is saved it is possible to choose either Streamed and Paged versions if the output format supports it.

Available datasources: None.

Figure 2.12. Report on Perspective



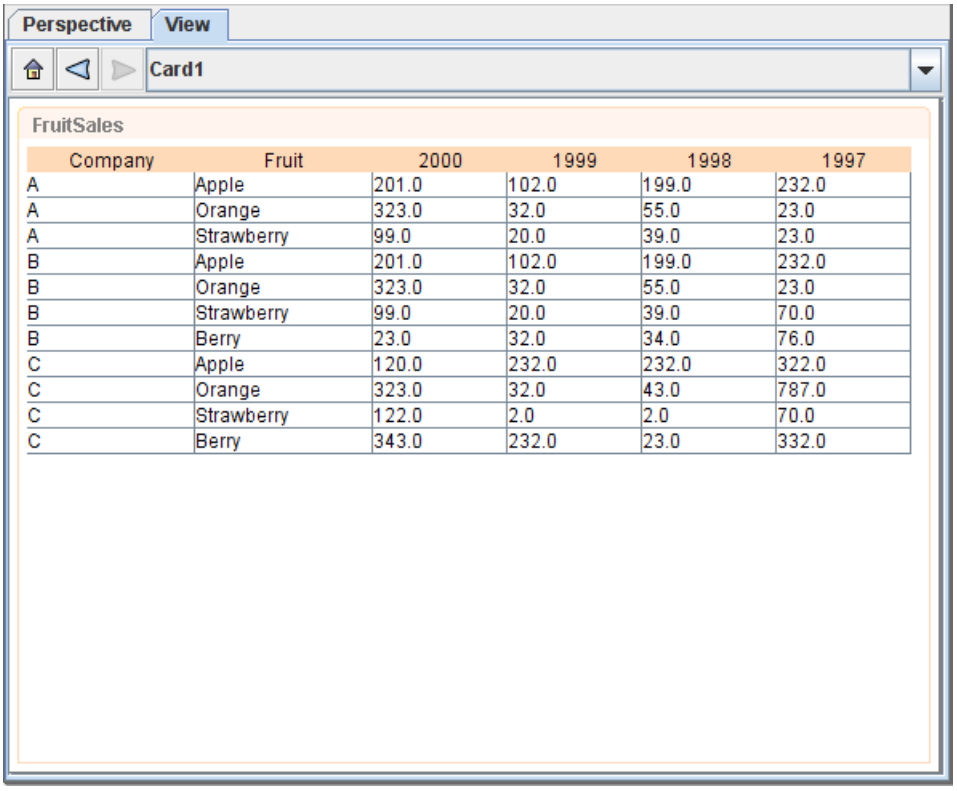
Table

A table shows records from a datasource, one row per record. This is a one-dimensional presentation, unlike the cube table which has two dimensions of variation. As with the other datasource-based contents, the records can be filtered before showing, and in the case of table, columns can also be discarded. This means the user can focus on just the few fields that are important. Of course, for even more control then table can be backed with a Composite DataSource that provides additional processing options.

Table data can be saved into a CSV, DataSource, Excel or XML file for subsequent processing.

Available datasources: All, Selected, Unselected.

Figure 2.13. Table on Perspective



Company	Fruit	2000	1999	1998	1997
A	Apple	201.0	102.0	199.0	232.0
A	Orange	323.0	32.0	55.0	23.0
A	Strawberry	99.0	20.0	39.0	23.0
B	Apple	201.0	102.0	199.0	232.0
B	Orange	323.0	32.0	55.0	23.0
B	Strawberry	99.0	20.0	39.0	70.0
B	Berry	23.0	32.0	34.0	76.0
C	Apple	120.0	232.0	232.0	322.0
C	Orange	323.0	32.0	43.0	787.0
C	Strawberry	122.0	2.0	2.0	70.0
C	Berry	343.0	232.0	23.0	332.0

Case Study 1

In this case study, filtering, sorting and extracting data from the wizard and using ${view:*ViewName.Selection*} will be covered. The idea of this dashboard is to select a *product_id* from a Grid element and a Table element will display records with the selected *product_id*. The border title for the Table element will display the *product_id* selected.

1. Create a JDBC datasource and name it *Sales* using MondrianFoodMart. Click on the `Query Builder` button and use the table named *sales_fact_1997*. Select all the fields. Infer the schema and end the wizard.

2. Create another JDBC datasource and name it *Sales_param*. Then, follow the steps mentioned above except in the SQL Query, enter *WHERE sales_fact_1997.product_id=${id}* after the line that says *FROM sales_fact_1997*. Infer the schema. When prompted for the id, enter *1* and exit from the wizard by clicking on the `Finish` button.

3. Create a new Perspective and give it a unique name. Drag Sales.ds to the perspective. Select `Add Grid`. When the wizard appears, click on the `Next` button twice. In the `Expression Builder`, enter 100 in the `Amount` field. Then, click on the `Set Value` button. This will extract the top 100 data from the datasource. Click on `Next` twice.

4. At this screen of the wizard, the user will need to select the field for the Grid element to display. In the drop-down list, select `product_id`. Finally, exit from the wizard by clicking on the `Finish` button.

5. The Grid element will be displayed on the perspective with `Sales` as the border title.

6. Drag Sales_Param.ds into the perspective. Place it on the right of the Grid element. On release of the mouse-click, select `Add Table`. At the second page of the wizard, you will see the parameter defined in the datasource. Under `Value`, change *${id}* to

${view:Sales.selected:product_id}. This will pass in the value selected by the user from the Grid element to Sales_param.ds as a parameter. The Table element will display the respective results.

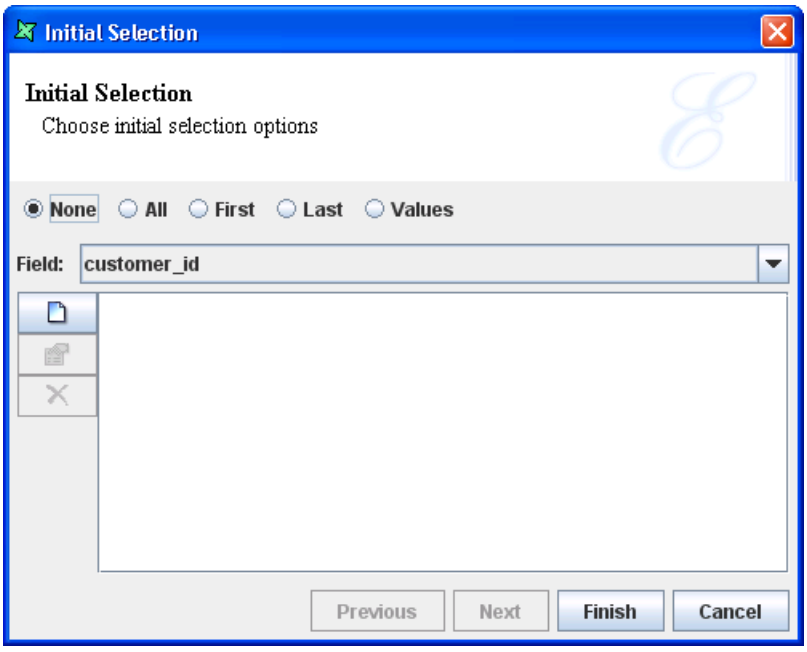
7. At the next page of the wizard, add a Sort by clicking on the `Add` icon. Select `product_id` and `Ascending` for the Name and Sort Order respectively. This will sort the data according to the *product_id* in ascending order.

8. Click on the `Next` button twice. Over here, you will be able to do some filtering of data. Under `When` of *promotion_id*, select *Not Equals*. As for the `Condition`, enter the value *0*. This will filter away all the records where the *promotion_id* is zero. Finally, click on `Finish` to exit from the wizard.

9. When an error appears saying that there is an error building the datasource table for repository caused by a syntax error in the query expression, this is the correct behavior. This is because when the perspective tries to load the data in all the elements, the Table element doesn't have any data to load. There is no value being passed to the parameter, therefore causing the error. To solve this problem, you can add `Initial Selection` to the Grid element. With `Initial Selection`, the Table element will have a value to pass to the datasource when it loads.

10. To add `Initial Selection`, right-click on the Grid element and select `Initial Selection...`. You will then see the Initial Selection Wizard like Figure 2.14, “Initial Selection Wizard”. Select `First`. For the `Field`, select *customer_id*. Click on `Finish`. With this setting,the first record of the Sales element will be selected (based on *customer_id*) when the dashboard first loads.

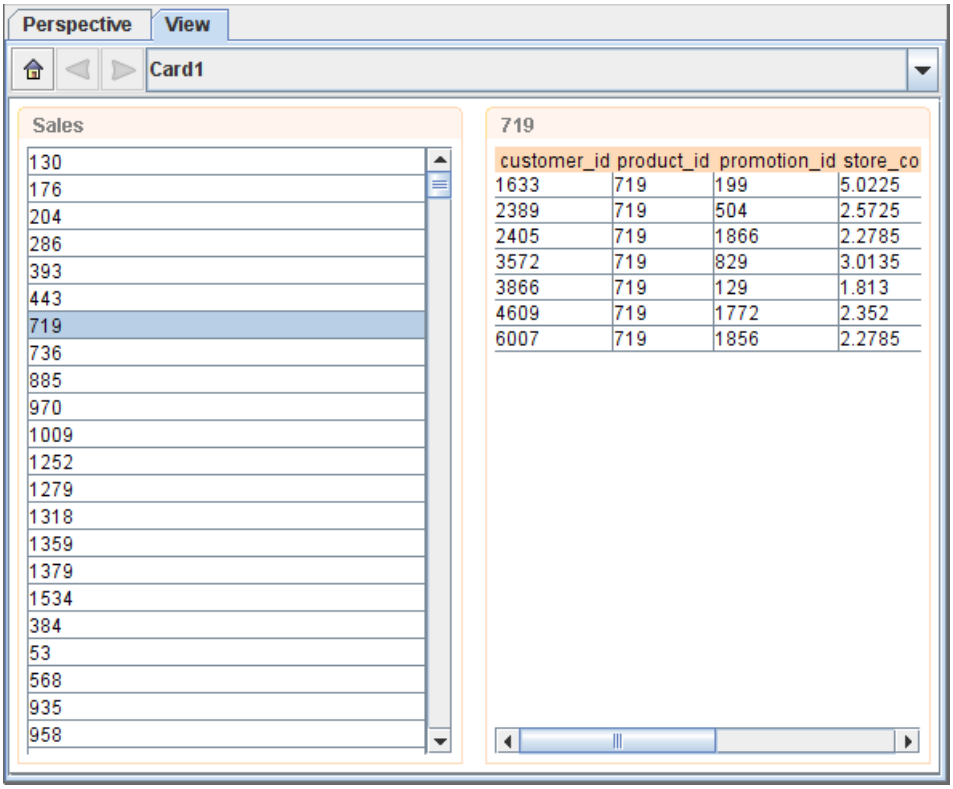
Figure 2.14. Initial Selection Wizard



11. In order for the border title of the Table element to display the *product_id* selected in the Grid element, right-click anyway on the Table element. Select `Presentation...`. The default value for Border Title is *${ContentName}*, which is the name of the element in the perspective. This name is editable at the properties of the element. To change it to reflect the selected *product_id*, enter *${view:Sales.selected:product_id}* in the Border Title field. Click on `Finish` to exit from the wizard.

12. Try clicking on some of the values under *Sales*. The value selected will also be the border title of the Table element, which is similar to Figure 2.15, “Results of Case Study”

Figure 2.15. Results of Case Study



Case Study 2

In this case study, we will demostrate how to use the Chart element drilldown function.

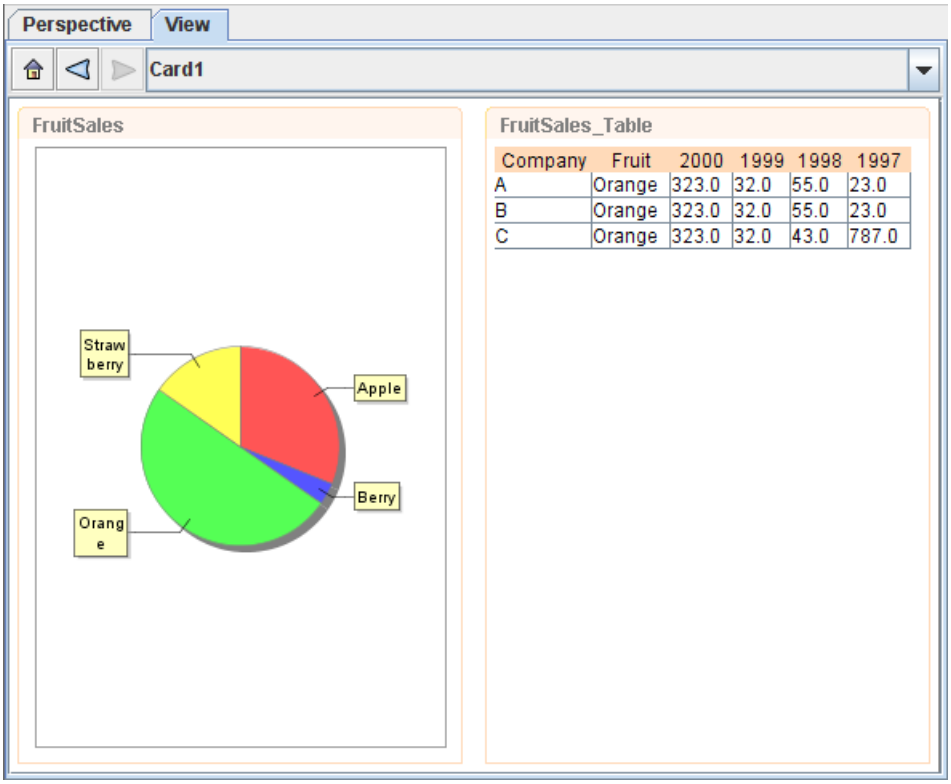
1. The first element to create, needless to say will be the Chart element. Drag FruitSales.ds into the Perspective and select `Add Chart`. After clicking on `Next` 4 times, you will need to select a chart type. Select *Pie* followed by *Pie chart* then `Next`.

2. For the `Key` tab, select *Fruit*. For `Value` tab, select *2000*. Exit the wizard by clicking on `Finish`.

3. Next, drag FruitSales.ds again into the Perspective and place it on the right. This time, select `Add Table`. On the first page of Add Content Wizard, click on the `View` tab and select *FruitSales* then `Selected`. Exit the wizard by clicking on `Finish`.

4. On the Perspective workspace, try clicking on the different slices on the Pie Chart. Since the Key of the Pie Chart is *Fruit*, when the user clicks on the different slice, the Table element will display the records with the selected *Fruit*, which is similar to Figure 2.16, “Results of Case Study 2”.

Figure 2.16. Results of Case Study 2



Chapter 3
Views

Introduction

Elixir Dashboard Designer provides a hierarchy of views to control the layout of content. The topmost view is the Card. There may be many cards in a dashboard, but only one card is visible at a time and switches between cards can be done either manually, using the combo box on the titlebar, or based on actions. For example, a double-click on a chart could switch to a card that contains a breakdown of the information in that chart.

Within a card, there may be two kinds of view, split views, which divide the available space amongst their children, and content views, which render content such as a chart or table.

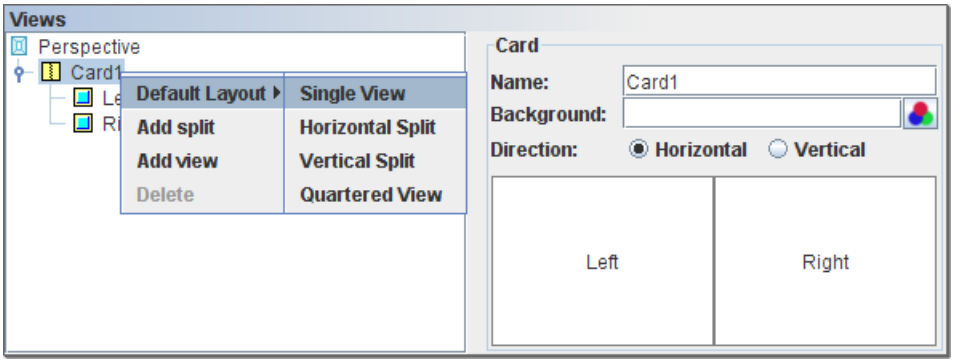
Card

The card is the top level view. As mentioned in the introduction a dashboard can contain many cards, but only one can be visible at a time. Every card must have a unique name, allowing it to be selected in response to user actions. A card is implemented as a kind of split view. So the discussion of split view below also applies to card. Cards can also define a background colour, which will fill the whole background of the dashboard. Individual contents can set their own background colours, which only affects their individual view rectangles. The default background colour is white.

Layout

When the Card is selected, the layout of the views will be displayed accordingly in the Card panel. Also, right-clicking on the Card then "Default Layout" shows options available for setting the layout of the Card, as shown in Figure 3.1, “Layout”. The name, background colour and direction of the Card can be changed from here.

Figure 3.1. Layout



Single View : A Single View, as seen from the name, will only 1 view can be seen.

Horizontal Split : A Horizontal Split will split the dashboard into 2 halves and each view placed beside each other. Only 2 views can be seen in the dashboard.

Vertical Split : A Vertical Split will split the dashboard into half and 1 view will be on top of the other. Like the Horizontal Split, only 2 views can be seen in the dashboard.

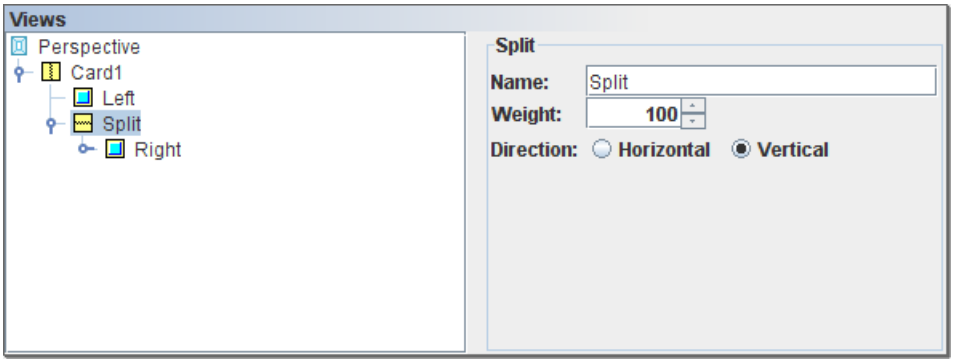
Quartered View : A Quartered View will split the dashboard into 4 equal squares.

After setting the Default Layout, a user can still add view(s) to the dashboard manually.

Split View

A split view will present it's child views (which may be content views or more splits) in a sequence, either left to right (horizontal mode) or top to bottom (vertical mode). The amount of space given to each child depends on the weight attribute of the child. Each child is given a space proportional to it's weight divided by the sum of all child weights. The default weight is 100, and unless changed will give equal spacing to each child. Weights are used rather than explicit sizes, in pixels or inches etc. so that the Perspective output can scale properly to be shown on different size devices. Editing can be done in the screen as shown below.

Figure 3.2. Split View



Splits can also be resized manually by dragging the gap between views. Resizing varies the weights of the cells on either side of the gap being dragged. This is a useful option for direct manipulation of the views themselves, in conjunction with drag and drop described in the section called “Drag and Drop”.

Content View

Each content view identifies a content by name to be shown within it. The content can be changed dynamically by the user, explicitly choosing a new content, or through an action which can trigger the content change. Each content view should have a unique name so that it can be referenced as part of a `view:` URL to link records to other contents.

Maximize

A single content view can be maximized to fill the entire Perspective. The view can be maximized by choosing the Maximize option from the titlebar, or by double-clicking on the titlebar. Similarly the view can be restored to it's original size by choosing the Restore option from the titlebar, or by double-clicking again on the titlebar.

It is also possible to maximize and restore a view with a user action, such as double-clicking. In this case, the view to be maximized might not even be on the current card. In this situation, the chosen view will be maximized as expected, and on restoration the original card will remain visible. This approach means it is easy to use other cards to hold a number of "popup" views and maximize them based on different user actions. Actions can also be used to restore views to their normal size, which has the effect of returning to the original card.

Drag and Drop

As well as populating cards using the wizard, it is possible to create views dynamically. Dragging and dropping glint files, report templates and datasources from the repository onto a card enables interactive positioning. With this approach, the tool will automatically insert the split views necessary to support the content views that are dropped.

When dragging over a view, a blue box is shown to indicate the current drop region. When near to the edge of a view, any drop will create a new content view on that side, introducing a split view if necessary. When dropping directly over a view, the old view will be replaced by a new one showing the new content. Note that when creating views this way, the content and view names are initialized automatically by the tool to ensure they are unique. It is possible to adjust the names later if desired.

Chapter 4
Actions

Introduction

Many kinds of content can fire events when the user interacts with them. These events may in turn trigger actions that manipulate or update the views being shown. This chapter describes the events that each kind of content can generate, what kinds of actions can be performed and then explores how to handle those events in order to trigger actions.

Content Events

Users interacting with content generate events that consist of three pieces of information <content,view,event>. The content attribute is the name of the content that the event occurred on. The view attribute is the name of the view the event occurred on. The event attribute is the name of the event. An example event might be: <Employees,LeftTable,doubleClick>. In this section we will describe the events that each kind of content can generate.

Chart	click:{key}[:{value}], doubleClick:{key}[:{value}], refresh, select
Cube Table	click, doubleClick, refresh, select
Glint	click[:{url}], doubleClick[:{url}], refresh
Report	click[:{url}], doubleClick[:{url}], refresh
Table	refresh, select
Grid	click:{key}[:{value}], doubleClick:{key}[:{value}], refresh, select

Elements written in braces like this: {xxx} are substituted by dynamic values depending on where the event occurs. These are usually preceded by a colon symbol (:). Elements written in square brackets like this [yyy] are optional, and may not be present if the information is not available. For example, a bar chart will have both a key and value string available, whereas a pie chart will only offer a key.

Here's a description of when each event is sent:

click	A click event is sent whenever a user clicks on a supporting view. The click may include extra information after the event name, for example click:{url} if the click occurs over a mouse-sensitive region in a glint or report, or click:{key}[:{value}] if the click occurs over a chart.
doubleClick	A doubleClick event is sent whenever a user clicks twice on the same view within a short preiod of time and without moving the mouse. A click event will always precede a doubleClick event.
refresh	A refresh event is sent each time a view is rendered because the underlying data has changed. It is not necessary to respond to this event to update dependent views, this is handled internally.
select	A select action occurs when a user changes the selection state of those contents that maintain a selection (eg. table). It is not necessary to respond to this event to update

dependent views (eg. those that depend on view:XXX.selected), this is handled internally.

Available Actions

When an event is received, it can trigger zero or more actions. The available actions are:

Maximize Action

The maximize action controls the maximized state of a named view. There are three types of action possible: Maximize, Restore and Toggle. Note that it is possible to maximize a view from a different card and it will show over the current card until it is restored or toggled off. The current card will not be affected by the action.

Refresh Action

The refresh action forces a named view to render it's content. The refresh action is usually handled internally for dependent views, but may be useful if the data is changed from outside Perspective, for example through JavaScript or because of a timed change.

Show Action

The show action replaces the current content of a named view with a different content.

Show Card Action

The show card action shows a named card in place of the current one.

Reset Action

The Reset action reverts the dashboard back to the default contents and initial selections. This action is particularly useful for web-based dashboards where the server session remembers the current state throughout the user session. Add a button image with a Reset action to your dashboard so that the user can easily reset the dashboard to its original state without having to logoff and logon again.

Render Report Action

The Render Report action will render the selected report in the perspective with the renderer selected by the user. The output is identical to the rendering in the Report Designer. Often dashboards will be created so that the user can dynamically construct the report contents by navigating through the data views. You can then provide a button image so that the final report can be preserved.

Generate Data Action

The Generate Data action can generate data in the selected element to 4 different formats. They are XML, Excel, Comma Separated Data (CSV) and Datasource. This action outputs the data as records, so any formatting of the display is not preserved. If you need to preserve the format of a dashboard cube then use the Generate Excel Cube Action below.

Generate Excel Cube Action

The Generate Excel Cube action can generate data in the selected cube as an Excel spreadsheet. Unlike the Generate Data Action, this action preserves the cube structure and cell rendering that you see on the dashboard.

Script Action

The Script action is the most powerful, because it can interact with the Perspective views, contents and even actions directly. This action is intended for power users and developers.

The JavaScript code executes in the context of the Perspective application itself, which provides some useful helper functions to access content and views.

- Content getContent(contentName);
- Panel getPanel(panelName);
- void refresh(panelName);

In addition, some context objects are available:

- Parameters: Use Parameters.get("Name") to access any dynamic parameter values.
- State: The overall dashboard state - from here you can navigate and interact with all parts of the dashboard.
- thisState: The state on which the event was triggered
- thisEvent: The event (String) that was triggered

Note

The full Perspective scripting API is described in Chapter 5, *Script Reference* and the separate PML Object Model documentation.

Triggering Actions from Events

Actions are defined using the Actions table on the Perspective tab. Each action identifies the trigger conditions by defining the combination of <content,view,event> that it responds to. An empty value means that it matches all possible events. For example <CubeData,select> matches the select event on CubeData (the content) in any view. For event name matching, only the text string up to the first colon is matched. This allows <MyChart,MyView,click:SomeURL> to be matched by an action defining <MyView,click> - which means any click on MyView.

Only actions which are marked as enabled are eligible to be invoked in response to an event. This allows a subset of actions and events to be tested and verified without side-effects from others.

Multiple actions can be triggered by a single event, if they all match the event attributes. The order of invocation is based on the order of actions in the Actions table. Actions can be raised or lowered in the action list to control the order of execution.

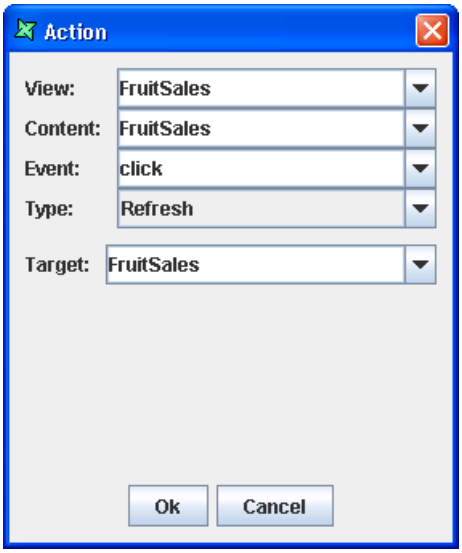
Case Study 1

This case study will do a simple illustration on how to use the `Generate Data` action.

1. Create a Perspective and give it a unique name.
2. Drag FruitSales.ds (available in /ElixirSamples/DataSource) into the perspective and select `Add Table`. Click through the wizard and click on `Finish` to exit from the wizard.
3. The Table element will be created on the perspective. To add an action to generate data, go to the `Perspective` tab. At the bottom of the screen, click on the `Action` tab, then click on the `Add` icon and the Action Wizard will appear, as shown in Figure 4.1, “Default screen of Action Wizard”.

With this action, on double-clicking on *FruitSales* (the Table element), you will be able to generate the data in *FruitSales*.

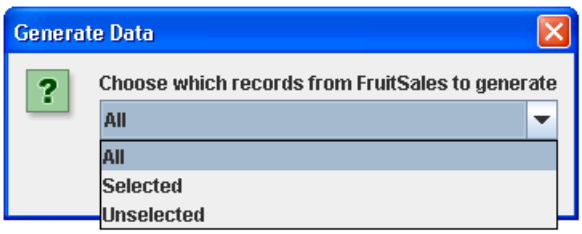
Figure 4.1. Default screen of Action Wizard



4. In the Action Wizard, you can customize the actions you want. In this case study, for `View`, select *FruitSales*. For `Content`, select *FruitSales*. For `Event`, select *doubleClick*. For `Type`, select *Generate Data*. For `View`, select *FruitSales*. Exit the wizard by clicking on the `Ok` button.

5. Go to the `View` tab. Try selecting a few records in *FruitSales* while holding on the *Ctrl* key. After selecting, double-click within the Table element and Generate Data Wizard will appear as shown in Figure 4.2, “Generate Data Wizard”. `All` will mean saving all the data in the Table element. `Selected` will mean saving only the selected records in the Table element. `Unselected` will mean saving all the records other than the selected ones.

Figure 4.2. Generate Data Wizard



6. Clicking on the `OK` button will lead you to the page for selecting the format you wish to save to, designate the location for saving and the name of the file. Clicking on `Save` will generate the file.

Case Study 2

In this case study, we will demostrate how to render a report in a Perspective which is similar to the Report Designer using Action. We can continue using the Perspective created in Case Study 1.

1. Since we are rendering a report, we will need a report template. Create a new report template and give it a unique name. In this case study, we name it *Fruit Report*. Click `Next`. For the datasource, we will use FruitSales.ds (available in /ElixirSamples/DataSource). Click `Next`. As for the report

type, we will use a Standard Report, Tabular Report. Click `Next`. For simplicity, we will use all the fields in FruitSales.ds. To move the fields to the *Report* column all at one go, you can click on the second button from the top. Click on `Finish` to exit from the wizard.

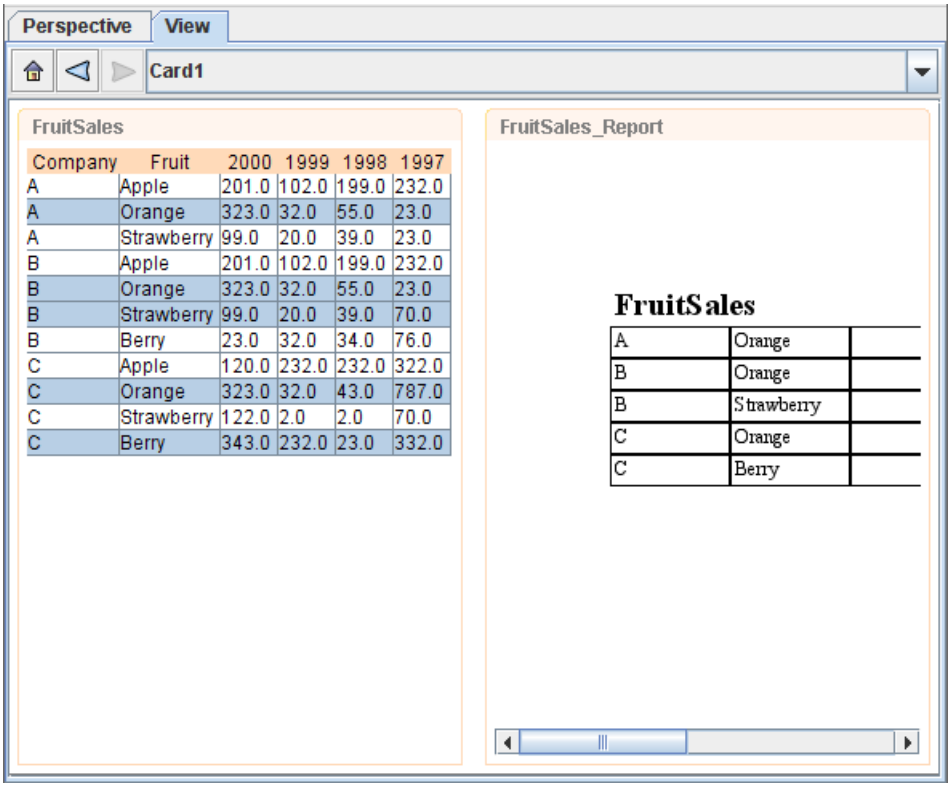
2. On exiting of the wizard, the report template would have been created. To add this report template to the perspective, go to the perspective and simply drag the file to the perspective. In this case study, we will add it to the right of *FruitSales*.

3. At the second page of the `Edit Content` wizard, click on the empty field under `Mapping`, then click on the `...` button. In the `DataSet Chooser` wizard, click on the `View` tab. Select *FruitSales*, then `Selected`. Exit the `DataSet Chooser` wizard by clicking on `Ok`. Exit the `Edit Content` wizard by clicking on `Finish`.

4. To add an action to generate the report template, go to the `Perspective` tab. At the `Action` tab, click on the `Add` icon and the Action Wizard will appear. For the `View` and `Content`, select *Fruit Report*. For the `Event`, select *doubleClick*. For the `Type`, select *Render Report*. For the `View`, select *Fruit Report*. Exit the wizard by clicking on the `Ok` button. With this, the report template will display records selected from *FruitSales*.

5. Now, try selecting some records from *FruitSales*. The report template will show the selected records, which is something like Figure 4.3, “Sample”.

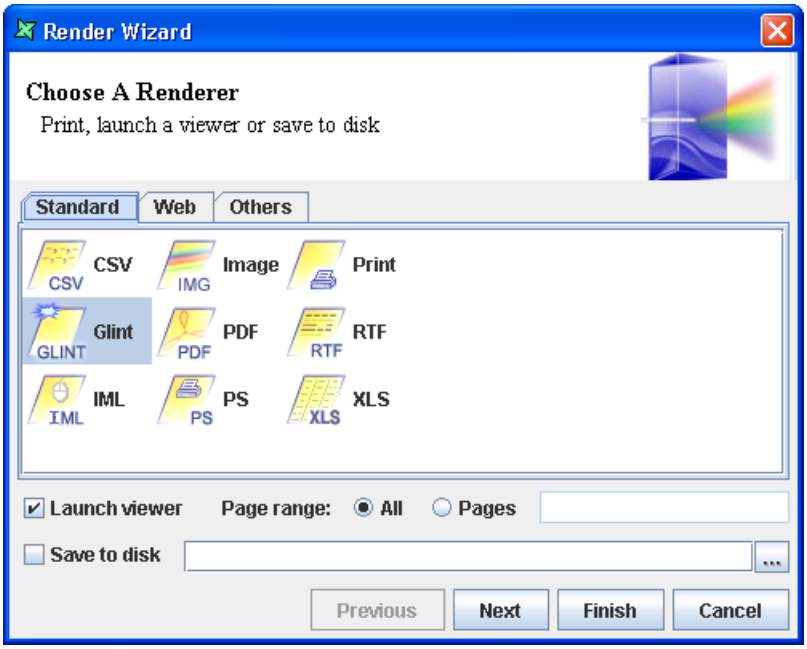
Figure 4.3. Sample



Company	Fruit	2000	1999	1998	1997
A	Apple	201.0	102.0	199.0	232.0
A	Orange	323.0	32.0	55.0	23.0
A	Strawberry	99.0	20.0	39.0	23.0
B	Apple	201.0	102.0	199.0	232.0
B	Orange	323.0	32.0	55.0	23.0
B	Strawberry	99.0	20.0	39.0	70.0
B	Berry	23.0	32.0	34.0	76.0
C	Apple	120.0	232.0	232.0	322.0
C	Orange	323.0	32.0	43.0	787.0
C	Strawberry	122.0	2.0	2.0	70.0
C	Berry	343.0	232.0	23.0	332.0

6. To render the report, double-click within the area of the report template (as configured in the Action wizard). The Render Wizard which is similar to the one in Report Designer will appear, as shown in Figure 4.4, “Render Wizard”. Select the format you wish to render the report to and run through the wizard if you need to do any extra configurations.

Figure 4.4. Render Wizard



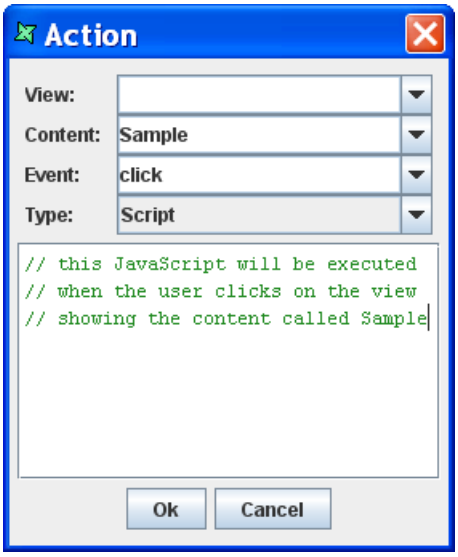
7. The report will be generated to your desired format after clicking on `Finish`.

Chapter 5
Script Reference

Introduction

Dashboards are scriptable using JavaScript Script Actions. This chapter will introduce the essential APIs for controlling the Dashboard through scripts. Each view that is positioned on a card in the dashboard is accessible through a state object. The view itself is not accessible, because it may be a graphical view (ie. in the Designer or Remote tool) or it may be an HTML view in a browser. The states can be identified by the name of the view they represent. To create a JavaScript action, choose the Perspective tab on the designer and add a new Action to the Action table at the bottom right. Choose Script as your Action type and proceed to add your script in the available field. The dialog is shown in Figure 5.1, “Script Action”

Figure 5.1. Script Action



Parameters

The Parameters object holds the dynamic parameters the user has entered.

- String get(String name)

This is equivalent to using ${name} in other parts of the dashboard. Within scripts ${substitution} will work, but Parameters.get("name") should be preferred because it is more efficient.

State

The State object represents the dashboard as a whole. It provides a set of useful services and acts as a container for the named states which link the contents and views that you define. To access a particular state, you need to append the name to the State object. For example, if you have a view named "MyCube", you can access the state of that view like this:

```
State.MyCube
```

If your view name contains spaces, then you need to use an alternate JavaScript syntax:

```
State.["My Cube"]
```

to achieve the same effect.

We will start by reviewing the API of the State object itself. Note that most of these services are available through the Action mechanism, so you don't have to use scripts to perform these commands - these services are only for situations where you want to chain a few actions together in a certain order, or maybe add some additional scripted logic.

void excelCube(String cubeName)	Generate the presentation of a cube as an Excel spreadsheet. The user will be prompted to enter a filename.
void generate(String contentName)	Generate the data shown by a particular content as a file. The user will be prompted to enter a filename and output type (eg. XML, CSV, XLS etc.)
String getCurrentCard()	Get the name of the current card
String getParameterValue(String name)	Get the value associated with the parameter name, or null if the parameter is not found.
Perspective getPerspective()	Get the Perspective object which represents the whole dashboard. See the PML Object Model documentation for full details on the dashboard object model.
IState getState(String name)	This provides another mechanism for getting the individual state objects, it is synonymous with State.name as described above.
void maximize(String viewName)	Maximize the named view, which need not be on the current card. The current card does not change, on restore the original card will show again.
void render(String reportName)	Render the named report. The user will be prompted to enter the render type and destination.
void reset()	Reset the current dashboard state back to the initial default view states and selections.
void restore()	Restore any maximized view back to the normal position.
void show(String view, String content)	Show the named content in the named view.
void showCard(String perspective, String card)	Replaces the current card with a new one, which could come from the same dashboard file, or jump to another dashboard. The perspective parameter should contain the full URL of a pml file (eg. "repository:/User/jane/mydash.pml") or null, if the card is located in the current dashboard.

Individual States

All state objects support the refresh() service, to force a refresh of the view. In most cases this is handled automatically by the dashboard when it detects that the display needs to be updated. All state objects also provide a getName() service, which will return the name of the state. Further to this, states which support selection allow scripted access to the selection values as described below:

CubeState
- DataTable getDataSource()
- List<DataRecord> getSelectedRecords()
- BitSet getSelection()
- boolean hasSelection()
- void setSelection(BitSet sel)

GridState
- boolean allSelected()
- DataTable getDataSource()
- List<DataRecord> getSelectedRecords()
- String getSelectedValue()
- BitSet getSelection()
- String[] getValues()
- boolean hasSelection()
- void setSelection(BitSet sel)

TableState
- DataTable getDataSource()
- List<DataRecord> getSelectedRecords()
- BitSet getSelection()
- boolean hasSelection()
- void setSelection(BitSet sel)

The infrastructure objects exposed by these APIs are DataTable and DataRecord, DataSchema and DataType. These are all in the com.elixirtech.data2 package. The useful services of these classes include:

DataTable
- int getRecordCount()
- DataRecord getRecord(int idx)
- DataRecord[] getRecords()
- DataSchema getSchema()

DataRecord
- Object getData(int idx)
- Object[] getData()
- DataSchema getSchema()

DataSchema
- int getColumnIndex(String columnName)
- String getColumnName(int idx)
- DataType getColumnType(int idx)

The following DataType constants are defined:

- BLOB

- BOOLEAN
- BYTE
- BYTE_ARRAY
- CHAR
- CLOB
- DATE
- DOUBLE
- FLOAT
- INTEGER
- LONG
- OBJECT
- SHORT
- STRING
- TIME
- TIMESTAMP
- UNKNOWN
- XML

Index

A
action, 2

C
card, 25
chart, 14
content, 2, 9, 26
csv, 15, 20
cube table, 15

D
datasource, 9, 15, 20
drag and drop, 27

E
events, 2
excel, 15, 20

F
filter, 9

G
glint, 15, 19
grid, 16

I
image, 18

M
maximize, 26

P
pdf, 19

R
report, 19

S
split, 26

T
table, 20
toolbar, 2

V
view, 2

X
xml, 15, 20